Crate: Technical Overview

Crate is an open source, highly scalable, shared-nothing distributed SQL database. Crate offers the scalability and performance of a modern No-SQL database with the power of SQL A Crate cluster is highly available and ready for production.

Crate's distributed SQL query engine lets you use the same syntax that already exists in your applications or integrations, and have queries seamlessly executed across the crate cluster, including any aggregations, if needed.

Crate is masterless and simple to install, operate and use. It handles transactional and analytical needs in one single database. Crate has been designed from the ground up to support the huge scale of Web, mobile and IoT applications. It is ideal in a microservices environment running Docker containers.

- Crate supports **nested structured data** with powerful **search** but also supports **binary objects** such as images and videos.
- Crate speaks **SQL**: the best language for querying and modifying data with an interface to existing tools such as ORMs. Crate combines the well known interface of SQL databases with the document oriented approach of modern NoSQL style databases.
- Crate is **super easy** to setup and very simple to administer. To grow Crate, just add hardware or cloud instances; it will do the rest. There is no need for manual sharding, partitioning, indexing or other complex administrative tasks to speed up queries.
- Crate is **simple and cheap** to operate: in a Crate cluster all nodes are identical and highly available, with automatic recovery on node failure, enabling the use of commodity hardware, virtual machines or containers.
- Crate is ideal for fast **growing, changing** and data intensive applications. All data is being sharded by default and can be easily partitioned, so it's easy to start with a small cluster and then grow to support huge datasets. By default, Crate indexes all fields and makes it possible to run arbitrary queries. This is very useful if requirements change. Most other datastore are inflexible to handle such changes.
- Crate is also ideal for **analytics** use-cases since at its heart it is a column based storage like Vertica, with caches for aggregation and sorting. Crate strongly supports the convergence of operational and analytics databases.
- Crate supports changes, simple and seamlessly. Schemas in Crate are dynamic, there are no locks needed on tables in order to add new columns or even nested objects.
- If you are looking for a traditional full ACID compliant relational database and need transactions you should not use Crate. However, most of these use cases can be still be implemented without transactions (and being more scalable) using concepts such as Optimistic Concurrency Control OCC, nested objects, denormalization or other alternatives instead of plain normalized relations and transactions.

Some Key Technical Aspects

- Crate server components are written entirely in Java and run on Java 7 or higher JVMs. Every machine in a Crate cluster is identical and has a single Java process that handles all required aspects. There are no special nodes required in order to run Crate. This is a key aspect of Crate's simplicity, because it makes it masterless and therefore horizontally scalable - you only need to add hardware to grow it. All data in the cluster will be balanced automatically when adding or removing nodes.
- Crate starts a Netty server and exposes its SQL-API via a HTTP-REST-Interface and a special high performance transport protocol. The transport protocol is used for node-to-node communication but also communicates directly with the native java client. By using Netty, any network communication is asynchronous, non-blocking and event driven. This is key for Crate's massive parallelism and high performance. Tables are always sharded and evenly distributed based on the hash of one or more columns. The routing is a function of the row and the number of configured shards of its table. This makes a central routing table unneccesary.
- Each shard is a full-blown Lucene index. Queries run on each required shard in parallel and are merged together either directly or by through re-distribution to intermediate reducers across the cluster. A planner defines how each query will be executed. This is set depending on data locality and query strategy. Field caches and other Lucene/ElasticSearch components are used to provide fast grouping and ordering support.
- Every shard is replicated by default. Lucene only appends data to files and never mutates them in place, which makes it predestined for replication and relocation use cases.
- Meta data such as table schemas and configuration is held in the cluster state, which is always in sync on all nodes all the time. Crate maintains a single auto-elected meta data master, which has authority in case of conflicting meta data updates. If the meta data master fails any other node in the cluster gets automatically elected.
- BLOBs in Crate are stored as immutable files and are sharded and replicated just like any other table data. Instead of Lucene indices the filesystem is used directly on blob shards. Any filesystem can be used for data directories. Crate stores each BLOB in one row with its unique hash. Metadata of BLOBs are stored in a normal table and referenced through the hash key. Storage and Consistency

Storage and Consistency

In this section we'll describe how Crate stores and distributes state across the cluster and the consistency and durability guarantees in Crate.

Data Storage

Reading data:
Every table in Crate is sharded, which means that tables are divided and distributed across the nodes of a cluster. Each shard in Crate is a Lucene index which is broken down into segments that are stored on the filesystem. Physically, the files reside under one of the configured data directories of a node.

Lucene only appends data to segment files, which means that data written to the disc will never be mutated. This makes it easy for replication and recovery, since syncing a shard is simply a matter of fetching data from a specific marker.

An arbitrary number of replica shards can be configured per table. Every operational replica holds a full synchronized copy of the primary shard.

In terms of read operations, there is no difference between executing the operation on the primary shard or on any of the replicas. Crate randomly assigns a shard when routing an operation. However it is possible to configure this behavior if required - for example in a Multi Zone Setup.

Writing data:

Write operations are handled differently than reads. Such operations are synchronous over all active replicas with the following flow:

1. The primary shard and the active replicas are looked up in the cluster state for the given operation. The primary shard and a quorum of the configured replicas need to be available for this step to succeed.

2. The operation is routed to the according primary shard for execution.

3. The operation gets executed on the primary shard

4. If the operation succeeds on the primary shard, the operation gets executed on all replicas in parallel.

5. After all replica operations are finished the operation result gets returned to the caller.

6. Should any replica shard fail to write the data or times-out in step 5, it is immediately considered as unavailable.

Atomic Operations on the Document Level

Each row of a table in Crate is a semi-structured document which can be deep-nested arbitrarily through the use of object and array types.

Operations on documents are atomic. This means that a write operation on a document either succeeds as a whole or has no effect at all. This is always the case, regardless of the nesting depth or size of the document.

Crate voted for maximum speed and to be eventual consistent instead of being ACID compliant. We believe the fast majority of use-case can be worked around transaction, in favor of agility and speed. Crate provides “transactional semantics” to work around most ACID requirements: every document in Crate has a

version number assigned to it. The numbers grows each and every time a change occurs. As a result, optimistic concurrency control exists and can greatly assist in working around the non-ACID non transactional nature of Crate.

Durability

Each shard has a WAL also known as translog. It guarantees that operations on documents are persisted to disk without having to issue a Lucene-commit for every write operation. When the translog gets flushed all data is written to the persistent index storage of Lucene and the translog gets cleared.

In case of an unclean shutdown of a shard, the transactions in the translog are replayed upon restart to ensure that all executed operations are permanent.

The translog is also directly transferred when a newly allocated replica initializes itself from the primary shard. As a result, there is no need to flush segments to disc just for replica recovery purposes.

Document Addressing

Every document has an internal identifier (see _id). This identifier is derived by default from the primary key. Documents living in tables without a primary key are automatically assigned a unique auto-generated id when they are created.

Each document is routed by its routing key to one specific shard. By default this key is the value of the _id column. However this can be configured in the table schema (see Routing).

Although it is transparent to the user, Crate accesses documents in two ways:

- **get:** Direct access by identifier. This is only applicable if the routing key and the identifier can be computed from the given query specification. (e.g.: the full primary key is defined in the where clause). This is the most efficient way to access a document, since only a single shard gets accessed and only a simple index lookup on the _id field has to be done.
- **search:** Query by matching against fields of documents across all candidate shards of the table.

Consistency

Crate is eventually consistent for search operations. Search operations are performed on shared IndexReaders which provide (alongside additional functionality) caching and reverse lookup capabilities for shards.
An IndexReader is always bound to the Lucene segment it was started from, which means it has to be refreshed in order to see new changes. This is done at several pre-defined times, but can also be done manually (see Refresh). Therefore a search only sees a change if the relevant IndexReader was refreshed after that change occurred.

If a query specification results in a “get” operation, changes are visible immediately. This is achieved by looking up the document in the translog first, which will always have the most recent version of the document. This enables “update and fetch:”if a client updates a row and that row is looked up by its primary key after that update, the changes will always be visible, since the information will be retrieved directly from the translog.

Every replica shard is updated synchronously with its primary shard and always carries the same information. Therefore, in terms of consistenct, it does not matter if the primary or a replica shard is accessed. A refresh of the IndexReader is the only activity that affects consistency.

Cluster Meta Data

Cluster meta data is held in the so called “Cluster State”, which contains the following information:

- Tables schemas
- Primary and replica shard locations – basically mapping from shard number to the storage node
- Status of each shard, which tells if a shard is currently ready for use or has any other state like “ initializing”, “recovering” or cannot be assigned at all
- Information about discovered nodes and their status
- Configuration information

Every node has its own copy of the cluster state. However, only one node is allowed to change the cluster state at runtime. This node is called the “master” node and is auto-elected. The “master” node has no special configuration at all and any node in the cluster can be elected as a master. If the current master node goes down for some reason another one will get automatically re-elected.

In order to avoid a scenario where two masters are elected due to network partitioning it is required to define a quorum of nodes in which it is possible to elect a master. For details in how to do this and further information see Master Node Election.

Here an example flow for an “ALTER TABLE” statement which changes the schema of a table, explaining the flow of events for any cluster state change.

A node in the cluster receives the ALTER TABLE request.

1. The node sends out a request to the current master node to change the table definition.
2. The master node applies the changes locally to the cluster state and sends out a notification to all affected nodes about the change.
3. The nodes apply the change, so that they are now in sync with the master.
4. Each node may take some local action depending on the type of cluster state change.

Import/Export

Crate allows you to export your data in JSON format so you can easily access the values and process them as you see fit. This is implemented using the COPY FROM/ COPY TO SQL statement. Each node will output JSON files for all of the primary shards it is aware of. It is also possible to copy data directly to/from Amazon S3 besides the local filesystem of each involved node.

Backup/Restore

Crate supports incremental backup and restore suitable for production environments (no lock of cluster during backup). It is possible to backup individual partitions, table or the cluster. Repositories can be created on HFDS, S3 etc to store incremental snapshots.